**Rapport de projet**

# Continuation Projet

# Bataille navale

**LEROY Florent**

Rapport rédigé pour

*Université de Rouen Normandie*

Informatique

Département d'informatique

juin 2024

# Table des matières

# 1 Introduction

## 1.1 Cahier des charges

L'**objectif** de ce projet est de réaliser un programme possédant une application graphique permettant à un ou deux joueurs de jouer au jeu 'Bataille navale'. Si un seul joueur est présent, il pourra jouer contre une intelligence artificielle dont il pourra choisir le niveau. Sinon, les deux joueurs pourront jouer en réseau.

L'exécutable devra être livré en format `.jar`. Et sera exécutable avec un simple script shell pour éviter tous problèmes.

Le programme proposera les fonctionnalités suivantes :

- Il sera possible de jouer contre un autre joueur, présent sur une autre machine (ou la même machine, en local). Dans ce cas-là, le programme utilisera le principe du modèle client-serveur afin de permettre à un joueur d'héberger la partie. Plus d'informations sur ce point seront disponibles dans la partie **Architecture réseau**.

- Sinon, le joueur pourra jouer contre une intelligence artificielle avec trois niveaux de difficulté disponibles (Facile, moyen et Difficile). Plus d'informations sur le comportement de l'intelligence seront disponibles dans la partie **Intelligence artificielle**.

- L'utilisation du programme se veut simple et intuitive. La manipulation des éléments graphiques se fera soit par simples clics sur des boutons, dans le cas du placement de bateaux sur la grille avant le début de la partie, la manipulation se fera par 'drag-and-drop' des éléments graphiques sur une zone cible. Certaines actions complémentaires, cependant, pourront nécessiter l'utilisation du clavier. Plus d'informations se trouveront dans la partie **Utilisation de l'exécutable**.

Vis-à-vis des règles du jeu 'Bataille navale', nous nous sommes basés sur les règles du jeu 'Touché-Coulé', commercialisé par Hasbro. Les règles sont :

- Chaque joueur dispose d'une grille de dimension $10 \times 10$ – pouvant être numéroté de **1** à **10** dans la longueur et de **A** à **J** dans la largeur.

- Chaque joueur pourra aussi disposer d'une grille supplémentaire afin de noter les tentatives d'attaques.

- Chaque joueur dispose de 5 blocs ('bateaux') de largeur 1 et de longueur différente :
  - 1 bateau de longueur 5
  - 1 bateau de longueur 4
  - 2 bateaux de longueurs 3
  - 1 bateau de longueur 2

- Un joueur ne peut voir la grille de son adversaire. Il voit seulement les endroits qu'il a touchés et le résultat de ces touches si c'est raté ou touché.

- En début de partie, chaque joueur dispose ses bateaux. Dès que tous les blocs sont placés, la partie peut commencer. Les bateaux peuvent être placés n'importe où sur la grille, tant que ceux-ci ne sont pas ou n'ont pas une partie d'eux sur un autre bateau ou en-dehors de la grille.

- Au tour par tour, un joueur va donner une coordonnée à l'adversaire. L'adversaire devra annoncer si, sur sa grille, la coordonnée contient une partie d'un bateau. S'ensuivent 3 scénarios :
  - La coordonnée ne continent pas de bateau. Dans ce cas, l'adversaire déclarera **'Raté'**.
  - La coordonnée contient une partie d'un bateau. Dans ce cas, l'adversaire déclarera **'Touché'**.
  - La coordonnée contient une partie d'un bateau et toutes les autres parties dudit bateau ont déjà été touché. Dans ce cas, l'adversaire déclarera **'Touché-coulé'**.

- À chaque tentative, le joueur devra noter sur sa grille auxiliaire le résultat du toucher.

- La partie se termine dès lors que chaque bateau d'un des joueurs ait été **'touché-coulé'**. Dans ce cas, l'adversaire sera proclamé vainqueur.

## 1.2 'Proof of concept'


FIGURE 1 – Concept de l'interface du jeu 'Bataille navale'

De gauche à droite, et de haut en bas :

- Interface en jeu contre un adversaire
- Interface de placement de bateaux
- Message d'indication en jeu
- Message de fin de partie

Cette maquette a été réalisée sous Figma, et a été créée avec des composants se voulant issus des directives et des modules de la bibliothèque GNOME/GTK4/LibAdwaita. La palette de couleur utilisée reprend la palette Catppuccin Mocha. Il s'agit seulement d'une vue d'artiste, n'ayant que pour objectif d'imaginer à quoi pourrait ressembler l'interface.

# 2 Version initiale

Ce projet étant la continuation du projet d'Application Informatique je vais rapidement expliquer l'architecture du programme ainsi que les choix faits afin de pouvoir mieux parler des modifications effectuées.

Le projet initiale dans un objectif d'utiliser un maximum les connaissances de deuxième et troisième année a été programmé en java 1.7 et les interfaces graphiques sont réalisées en java swing. Le serveur lui est programmé en python, car il ne travaille que sur des chaînes de caractères.

## 2.1 Architecture du programme

Dans le programme, il y a deux architectures à évoquer. L'architecture du code en lui-même et l'architecture réseau, la manière dont est utilisé le serveur. Pour expliquer l'architecture réseau, nous allons simuler un tour où le client 1 joue pour essayer de toucher un bateau du client 2.

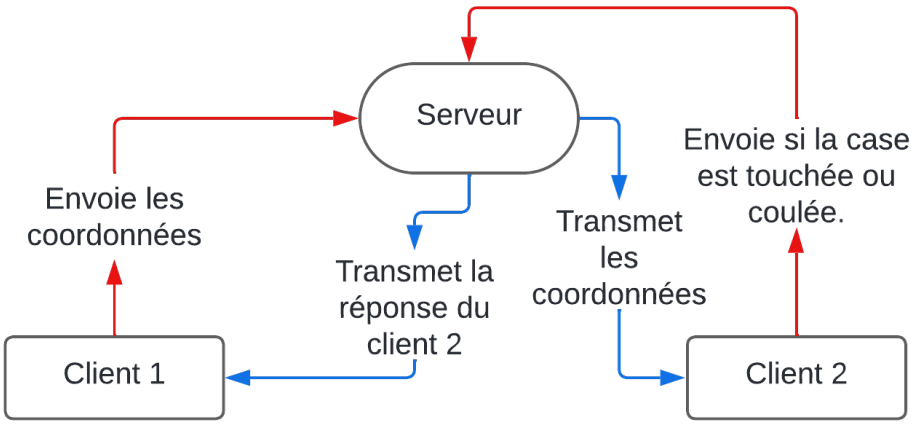


FIGURE 2 – Diagramme fonctionnel de l'architecture réseau lors d'un tour.

Via le diagramme on voit que le serveur ne sert que de proxy en transmettant simplement les informations d'un client vers un autre sans vérifier les informations.

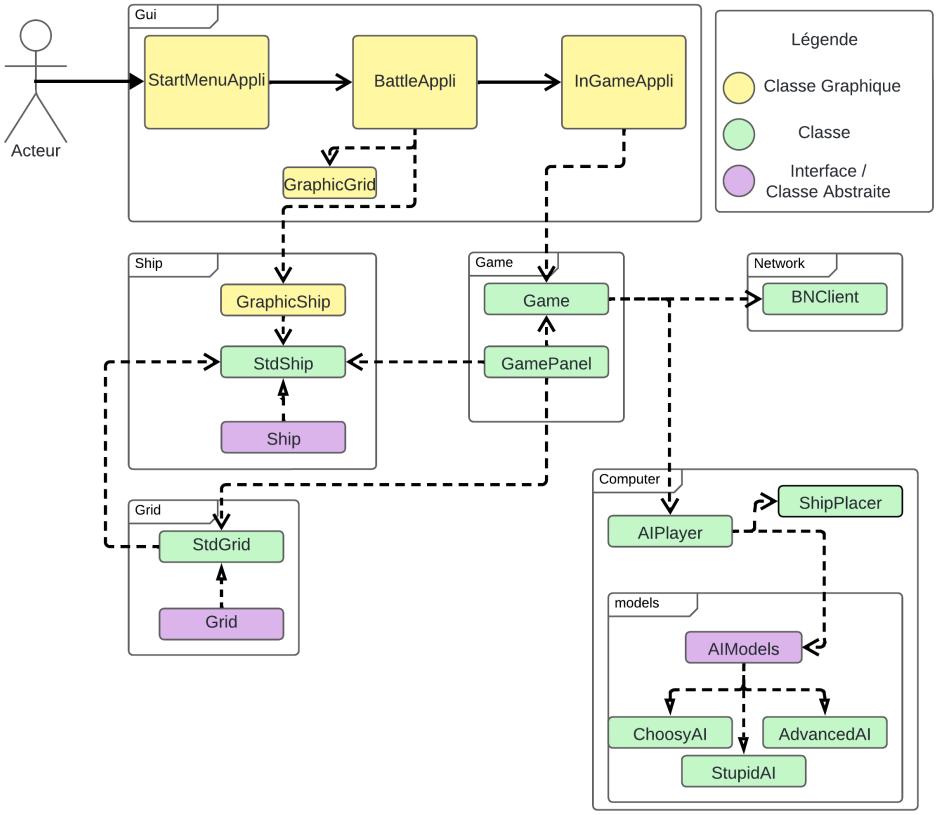


FIGURE 3 – Diagramme UML de la structure du programme

## 2.2 Apparence de la version initiale

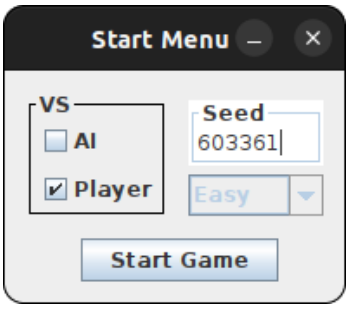


FIGURE 4 – Écran de début de partie

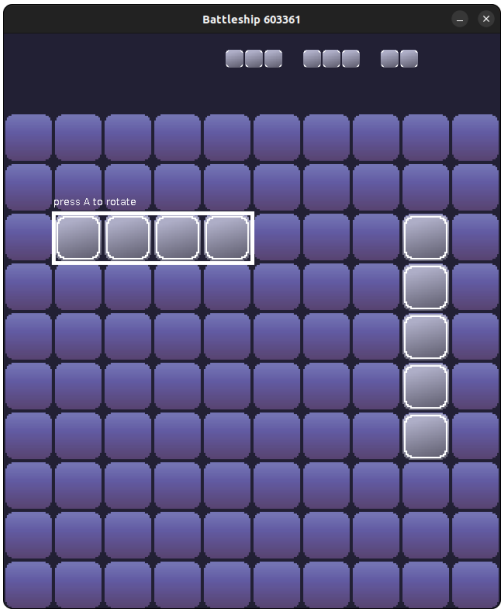


FIGURE 5 – Écran de placement des bateaux

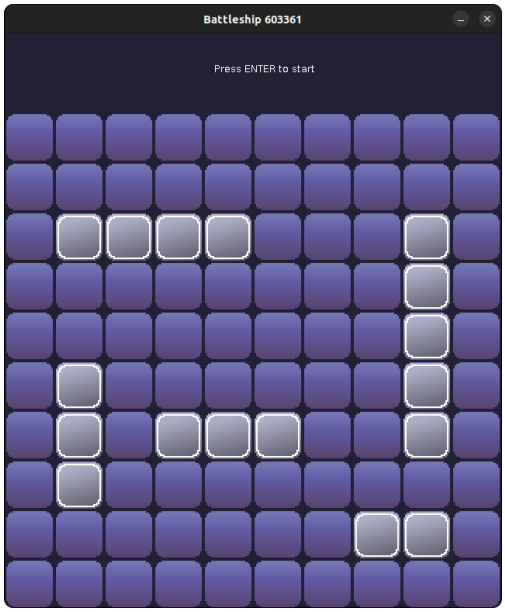


FIGURE 6 – Écran de placement des bateaux (après avoir placé tous les bateaux)

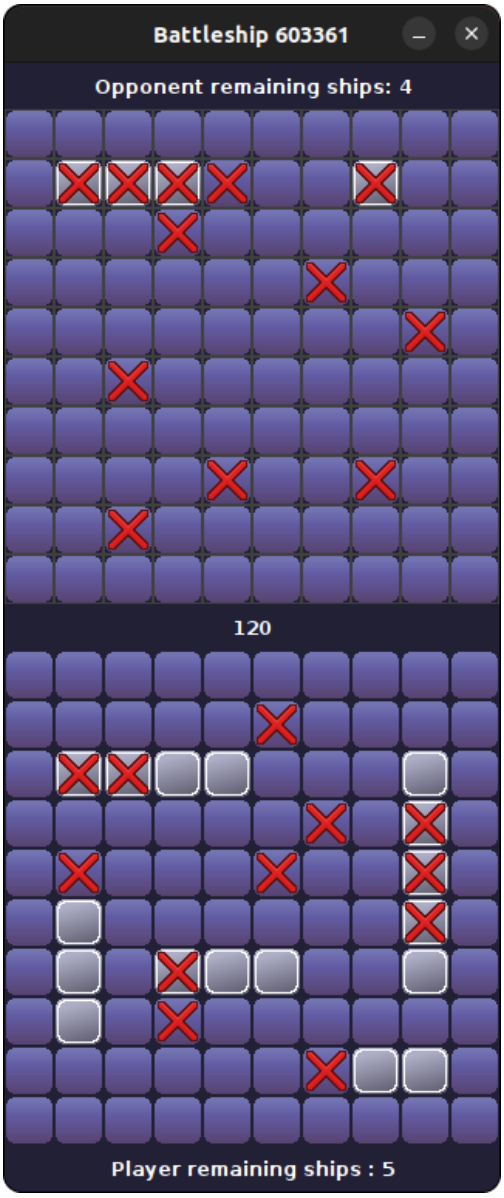


FIGURE 7 – Écran de partie (avec un autre joueur en ligne)

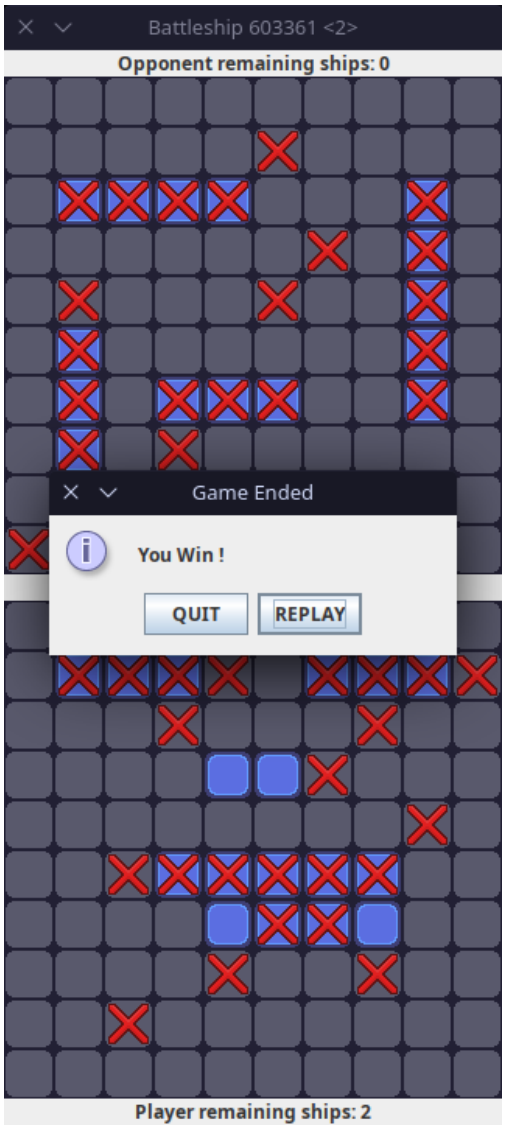


FIGURE 8 – Écran de fin de partie

## 2.3 Intelligence artificielle

L'implémentation de l'intelligence artificielle n'ayant pas été modifiée par rapport à la version initiale. Cette partie est à but de rappeler.

Afin de permettre à un joueur de jouer en absence d'adversaire humain ou en absence de connexion, nous avons implanté un ensemble d'algorithmes permettant de simuler le comportement d'un joueur.

### 2.3.1 Implémentations

Du point de vue de l'implémentation, la classe responsable de représenter et de permettre une interaction avec l'intelligence artificielle se nomme `AIPlayer`, et se présente comme une couche entre `GamePanel` et `Game` où s'effectue l'action de l'IA.

Ces méthodes permettent d'annoncer à la machine adverse où elle a été touchée, de toucher le joueur selon des coordonnées (voir ci-dessous), et de l'informer en retour si le tir a réussi ou pas.

Le plus grand du travail est réalisé par les classes réalisant la classe abstraite `AIModel`. Ces classes réalisent la méthode `chooseCoordinates ()` qui choisit de manière méthodique (ou non) l'emplacement du prochain coup.

### 2.3.2 Algorithmes

Dans notre implémentation actuelle, il existe 3 modèles d'IA différents, classés selon leur niveau de difficulté :

- `StupidAI` représente l'IA de difficulté facile. Son comportement est relativement simple : à chaque tour, l'IA récupère les coordonnées de toutes les cases non touchées et choisit une coordonnée au hasard.

- `AdvancedAI` représente l'IA de difficulté difficile. Son comportement est, lui aussi, relativement simple : à chaque tour, l'IA récupère les coordonnées de toutes les cases non touchées, et les coordonnées des cases autour des points touchés et réussi (nommés 'touchées' dans les règles du jeu), puis choisis en priorité les cases adjacentes aux coups réussis, si ces cases n'existent pas, il choisit une coordonnée au hasard sinon, revenant donc au comportement de l'IA de difficulté facile.

- `ChoosyAI` représente l'IA de difficulté intermédiaire. Son comportement est un mélange des comportements des deux IA indiqués ci-dessus, la seule différence résidant dans le fait que l'IA peut choisir soit une case au hasard avec une chance de $\frac{2}{5}$, soit une case adjacente, avec une chance de $\frac{3}{5}$. Deux macroconstantes peut être utilisée pour modifier ces probabilités.

### 2.3.3 Améliorations possibles

Il existe plusieurs moyens pour rendre l'intelligence artificielle plus efficace, et donc plus dure.

- Pour accélérer le repérage des bateaux, nous pouvons procéder avec un motif en carreaux, comme suit :

$$\begin{bmatrix} & X & \\ X & & X \\ & X & \end{bmatrix}$$

  Sachant qu'un bateau est de longueur minimum 2, cela signifie que l'on peut trouver au moins une des cases du bateau, quelque soit son emplacement.

$$\begin{bmatrix} & X & \\ X & & O \\ & X & O \end{bmatrix} \begin{bmatrix} & O & \\ X & O & X \\ & X & \end{bmatrix} \begin{bmatrix} O & O & \\ X & & X \\ & X & \end{bmatrix} \begin{bmatrix} & X & \\ X & & X \\ O & O & \end{bmatrix} \dots$$

  Cela signifie aussi que le nombre de cases potentiellement touchables sont au plus divisées par 2, rendant l'IA bien plus redoutable. Une fois un bateau trouvé, le comportement ne changera pas.

- Une des variables potentiellement accessible à l'IA pouvant avoir une influence sur son comportement est le nombre de bateaux restant, et surtout sa variation après avoir touché un bateau. Cela pourrait indiquer à l'intelligence artificielle qu'il faut qu'elle arrête de chercher autour de ce bateau, car celui-ci a déjà été touché-coulé.

  Nous pouvons remarquer que cette amélioration n'augmente pas réellement la difficulté de l'intelligence artificielle, sachant que ce comportement est assimilable à l'action de chercher des cases adjacentes aux bateaux déjà touchés.

### 2.3.4 Placement des bateaux

Pour permettre à l'IA de placer des bateaux sans intervention humaine, elle utilise la classe `ShipPlacer`, lui permettant de placer les bateaux de manière complètement aléatoire, et donc impossible à prédire par le joueur humain adverse. Cela se fait par l'appel à la méthode `placeShips ()`.

Pour ce faire, chaque bateau est placé dans l'ordre de génération (voir la méthode `generateShips ()`), avec une coordonnée de départ et une orientation (horizontale ou verticale) aléatoire. L'algorithme utilisé est de type glouton/brute force, c'est-à-dire que le prochain bateau ne sera placé qu'une fois que le bateau précédent a trouvé un emplacement adéquat, à savoir un emplacement ne dépassant pas de la grille et non occupé par un autre bateau.

Deux remarques sont à faire sur cet algorithme.

Tout d'abord, on peut se poser la question sur le temps d'exécution de cette méthode, voire sa complexité. En effet, la capacité de l'algorithme à se terminer dépend des nombres aléatoires tirés et du placement des bateaux précédents, donc sa complexité et son temps d'exécution est grandement variable.

Nous pouvons au moins assurer la capacité de l'algorithme à atteindre la fin. En effet, il n'existe, à notre connaissance, aucun placement de $n - 1$ bateaux (ici, $n = 5$) sur une grille de $10 \times 10$ (à savoir la grille utilisée pour la modélisation du jeu) qui empêcherait le placement du $n^e$ bateau, la grille proposant toujours suffisamment de cases et d'emplacements de taille maximum 5 pour placer le dernier bateau, quelle que soit sa taille.

La seconde remarque tient sur le fait que, malgré la capacité de l'algorithme à placer chaque bateau avec succès, quels que soient les nombres aléatoires tirés, la taille des bateaux et la relative petitesse de la grille peut forcer l'algorithme à toujours placer ses bateaux dans la même orientation, ce point étant à ce jour le seul comportement à peu près prévisible venant du placement des bateaux.

# 3 Nouvelle Version

Cette nouvelle version du projet a pour but d'appliquer les améliorations possibles que nous avons constatées pour le projet initial. Et aussi d'utiliser des outils récents pour le réaliser. C'est pour cette raison que cette version du projet a été programmée en utilisant Java 21 ainsi que JavaFX 21 et FXML pour les interfaces graphiques.

Pour cette nouvelle version j'ai aussi utilisé un gestionnaire de dépendance `Maven`. Cela a pour principal avantage que, par exemple moi et mon professeur référent n'utilisons pas le même IDE mais grâce à Maven nous n'avons pas eu de problème de compatibilités car Maven permet de gérer ce genre de chose.

## 3.1 Architecture du code coté client

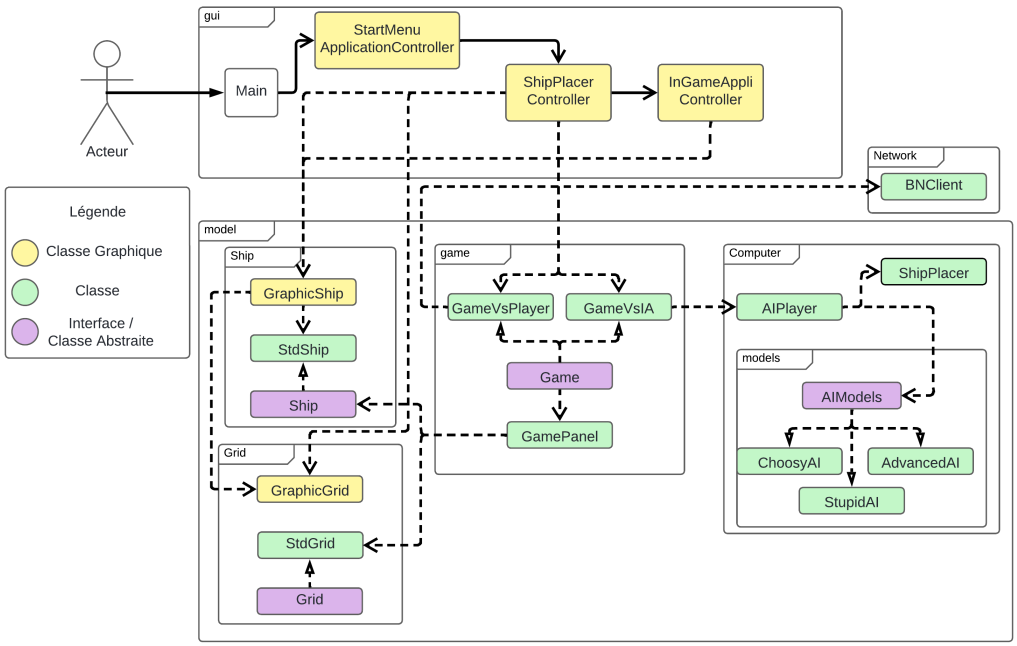


FIGURE 9 – Diagramme UML architecture code coté client

Maintenant, nous allons rentrer un peu plus dans le détail de chaque classe ou module afin de mieux expliquer le fonctionnement du programme.

### 3.1.1 Module `Ship`

Le module `Ship` représente les bateaux, il sert de modèle à la classe graphique utilisé pour la représentation et la manipulation via 'drag-and-drop' dans la classe `GraphicShip`. Il a pour but de retenir le type de chaque bateau (sa taille) ainsi que les coordonnées sur lesquelles il est positionné. La classe `Ship` n'est composée que de deux éléments :

- Sa longueur : la taille en nombre de cases occupées par le bateau. Taille passée en paramètre au constructeur lors de l'instanciation d'un nouveau bateau.

- Ses coordonnées : une liste de `Point2D` contenant les coordonnées de chaque case occupée par le bateau.

### 3.1.2 Classe `TileState`

La classe `TileState` est une classe énumérée représentant toutes les valeurs que peut avoir une case de la grille de bataille navale.

### 3.1.3 Module `Grid`

Le module `Grid` simule le comportement d'une grille de bataille navale. Lors de l'instanciation d'une grille, on lui donne en paramètre un tableau de `Ship` afin que la grille ait connaissance de l'emplacement de chaque bateau. Dans ce module, il y a une méthode principale :

```
    void hit (int row, int col);
```

Elle simule un tir sur la case de ligne `row` et colonne `col` cette fonction peut renvoie une exception `PropertyVetoException` si la condition associée au `VetoableChangeSupport` sur la propriété `PROP_GRID` via la fonction `addVetoableChangeListener` n'est pas respectée.

La méthode

```
    List<List<Point2D>> getShip ();
```

renvoie une liste où chaque élément de cette liste est une liste contenant les coordonnées des cases occupées par un bateau. Lorsqu'une case occupée par

un bateau est touchée, on retire cet élément de la liste associée au bateau lorsque la liste associée à un bateau est vide cela veut dire que le bateau est entièrement détruit donc nous retirons cette liste de la liste contenant les bateaux. Ainsi, si la liste est vide alors tous les bateaux de la grille ont été détruits. C'est pour cette raison qu'à chaque fois que l'on touche un bateau, on doit recalculer la liste des bateaux.

Les attributs statiques de type `String` dans l'interface `Grid` sont les noms des propriétés auxquelles on ajoutera des écouteurs qui seront notifiés à chaque modification de la valeur associée afin d'actualiser l'interface graphique.

### 3.1.4 GamePanel

La classe `GamePanel` s'occupe de la simulation de la partie du côté du joueur, elle gère donc la grille du joueur qui est une instance de `Grid` ainsi que la version 'cache' de la grille de l'adversaire. Cette version 'cache' de la grille de l'adversaire est la manière dont le joueur voit la grille son adversaire. Au début de la partie, elle est vide, lorsque le joueur tape l'ennemi, il voit sur cette version si la case sur laquelle il a tiré est 'touchée' ou 'ratée'. Il possède aussi un attribut représentant le nombre de bateaux restant à son adversaire. Tous comme le module `Grid` le module `GamePanel` possède des

attributs statiques de type `String` qui les noms des propriétés auxquelles on ajoutera des écouteurs qui seront notifiés à chaque modification de la valeur associée. Le module `GamePanel` possède deux méthodes principales :

```
void playerHitEnnemy (int row, int col, TileState value);
TileState ennemyHitPlayer (int row, int col);
```

- `playerHitEnnemy` met à jour la version 'cache' de la grille adverse en mettant la case de ligne `row` et colonne `col` à la valeur passer en paramètre.

- `ennemyHitPlayer` tire sur la grille du joueur à la case de ligne `row` et colonne `col` et renvoie le nouvel état de la case touchée.

### 3.1.5 BNClient

Gère la connexion avec le serveur ainsi que l'envoi des données et la réception de celle-ci. Afin d'en faire un module pouvant être facilement réutilisé juste en modifier la valeur des deux attributs statiques, il est considéré que le module ne connaît pas le protocole de communication avec le serveur et que seuls les modules utilisant `BNClient` en ont connaissance.

### 3.1.6 Game

La classe Game est la classe mère dans deux autres classe `GameVsPlayer` et `GameVsIA`. Elle regroupe les attributs ainsi que les méthodes communes aux deux autres classes.

```
void hit (int row, int col)
```

- La méthode `hit` a pour but d'être appelé par l'interface graphique lors du tour de joueur, signifie que le joueur essaye de toucher la case de ligne row et colonne col. L'implémentation de cela dépend uniquement de si la partie est contre un joueur ou une IA, mais, reste similaire dans le principe.

```
void getHit ()
```

- La méthode `getHit` est la version inverse de `hit`. Elle est appelée lorsque c'est le tour de l'adversaire et que le joueur attend que l'on lui dise quelle case est touché. Cette méthode est structurellement très différente selon le mode de jeux chaque, ces différences seront expliquées dans les parties ci-dessous consacrées à chaque type de partie.

```
void run ()
```

- La méthode `run` s'occupe de la gestion du timer lors d'un tour de la partie. La méthode pour annoncer le changement de tour diffère selon le mode de jeu, mais reste très similaire.

### 3.1.7 GameVsPlayer

Cette classe a pour but de gérer la partie cotée joueur lorsque que la partie est contre un autre joueur. Elle doit donc gérer toute la partie connexions au serveur, envoie des messages ainsi que la réception des messages du serveur et le traitement desdits messages (message d'erreur, réponse du serveur, arrêt de la connexion).

```
void getHit ()
```

- La méthode `getHit` dans la partie contre un joueur ne fait qu'attendre que le serveur lui dise quelle case est touché et quelle est la valeur de cette case. Et met à jour le modèle client ainsi que l'interface graphique en fonction des informations reçues.

### 3.1.8 GameVsIA

Cette classe gère la partie contre l'intelligence artificielle. Les écoutent et envoie sur le serveur sont remplacées par des appels aux fonctions du modèle de l'IA.

```
void getHit ()
```

- Contrairement à lorsque que la partie est contre un joueur, ici, c'est le programme client qui effectue les calcules et vérifications de si une case peut être touché et quelle est la valeur de cette nouvelle case. Puis communique avec l'intelligence artificielle sur les résultats de ses calculs.

```
void run ()
```

— Contre une IA la méthode `run` gère aussi le système de tour par tour, c'est elle qui dit au joueur et l'IA s'ils doivent jouer ou non.

## 3.2 Architecture réseau

Afin d'expliquer l'architecture réseau de cette nouvelle version, nous allons reprendre le même exemple que précédemment, c'est-à-dire simuler un tour où le client 1 joue pour essayer de toucher un bateau du client 2.

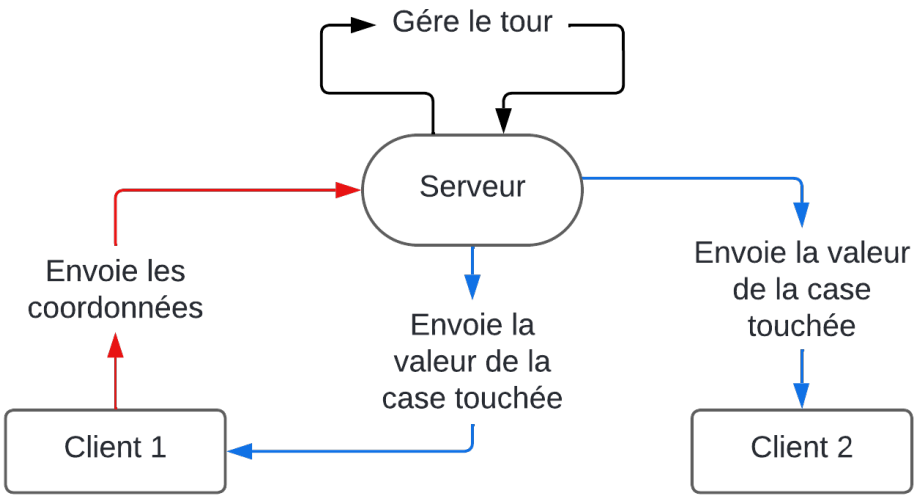


FIGURE 10 – Diagramme fonctionnel lors d'un tour Bataille Navale V2

Sur cette version le serveur n'est plus un simple proxy, c'est lui qui gère la partie, les clients eux ne sont que des affichages graphiques et n'envoie au serveur que des coordonnées. Pour plus de précision sur la gestion de la communication entre client et serveur voir la section **Serveur**.

## 3.3 Architecture code coté serveur

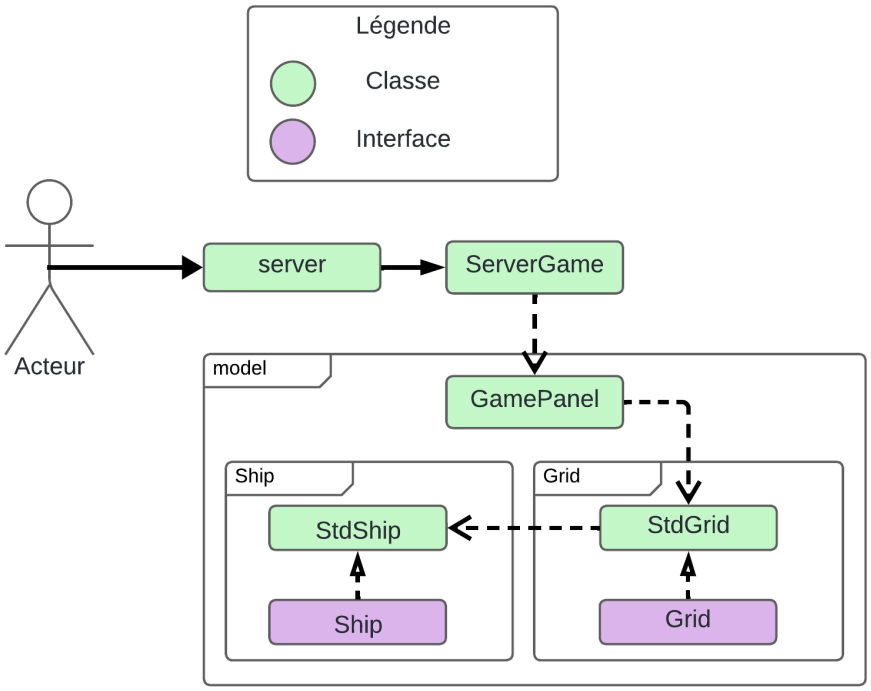


FIGURE 11 – Diagramme UML architecture code serveur

Les classes telles que

- GamePanel
- Ship et StdShip
- Grid et StdGrid

sont les mêmes que pour le programme client, mais sans aucune partie pour les interactions avec une interface graphique.

Les deux nouvelles classes sont `server` et `ServerGame`

- `server` est la classe que s'occupe des connexions des joueurs. Les client ce connecte au serveur via le port 6969, le serveur lui écoute sur ce port. Cette classe lie entre eux les clients ayant donné la même clé. Les deux sockets liés aux clients sont alors donnés comme argument au constructeur de la classe `ServerGame`.

- `ServerGame` est la classe qui gère la partie pour les deux joueurs. Elle possède les grilles des deux joueurs et gère le tour par tour. Lors d'un tour, elle attend l'envoi des coordonnées par le joueur dont c'est le tour puis envoie la réponse aux deux clients. C'est aussi cette classe qui annonce lorsque la partie est finie et qui est le joueur gagnant.

### 3.3.1 Communication entre serveur et clients

Nous allons maintenant évoquer la manière dont communique le serveur et les clients. Les communications se font uniquement via des chaînes de caractères afin de faciliter les échanges.

Le client peut envoyer des demandes avant même qu'une partie soit lancé et aussi pendant un tour. Ces demande sont :

- **SEP** : Demande du client pour connaître le séparateur utilisé dans les messages, il correspond à ' :' actuellement.

- **SEP_BETWEEN_SHIPS** : Demande du client pour connaître le séparateur entre les bateaux utilisé dans le message d'envoi des bateaux, il correspond à ' ;' actuellement.

- **SEP_BETWEEN_COORDS** : Demande du client pour connaître le séparateur entre les coordonnées d'un bateau utilisé dans le message d'envoi des bateaux, il correspond à ',' actuellement.

- **SEP_BETWEEN_XY** : Demande du client pour connaître le séparateur entre des coordonnées, il correspond à '-' actuellement.

Lorsque le message vient du client, il est de la forme :

**SEND + séparateur + Message**

et lorsqu'il vient du serveur, il est de la forme :

**S + séparateur + Message**

Je vais maintenant parler de l'ensemble des messages pouvant être envoyé entre le serveur et un client, voici donc la liste de ces valeurs ainsi que leur signification.

- **PENDING** :Message du serveur pour informer le client de l'attente d'une autre connexion pour démarrer la partie.
- **CONNECTED** : Message du serveur pour informer le client que la partie va commencer.
- **TURN_TIME** : Demande du client pour connaître la duré d'un tour.
- **PLAY** : C'est le tour du joueur que c'est son tour.
- **WAIT** : C'est le tour de l'autre joueur.
- **LOSE** : Le joueur à perdu la partie
- **WIN** : Le joueur a gagné la partie
- **CHANGE_TURN** : Message du client indiquant que son temps pour un tour est écoulé.
- **CANT_HIT** : La case choisie ne peut pas être touchée.
- **MISS** : La case touchée est une case vide.
- **HIT** : La case touchée est une case contenant un bateau.
- **SUNK** : La case touchée est une case contenant la dernière partie non touchée d'un bateau. Il y a donc un bateau de moins.

### 3.3.2 Message d'erreur lors de la communication

Tous les messages d'erreurs envoyés par le serveur sont de la forme.

**S + Séparateur + ERR + Séparateur + [CODE D'ERREUR]**

- **BADARGS** : Le message reçu n'est pas celui attendu ou ne respecte pas le pattern attendu.

- **BADCODE** : La clé de partie donnée par le client ne respecte pas le format attendu.
- **ALRDYCONNECTED** : Le client est déjà connecté à une partie.
- **ALRDYPENDING** :Le client attend déjà pour une partie.
- **DISCONNECTED** : Le client a été déconnecté du serveur dû à la déconnexion de l'autre client.

### 3.3.3 Envoie des coordonnées des bateaux au serveur

Comme dit précédemment les échanges clients serveur sont uniquement via une chaîne de caractères, il y a donc dû trouver un protocole pour envoyés les coordonnées des bateaux placées par chaque client.

Pour expliquer cela, nous allons prendre un exemple.

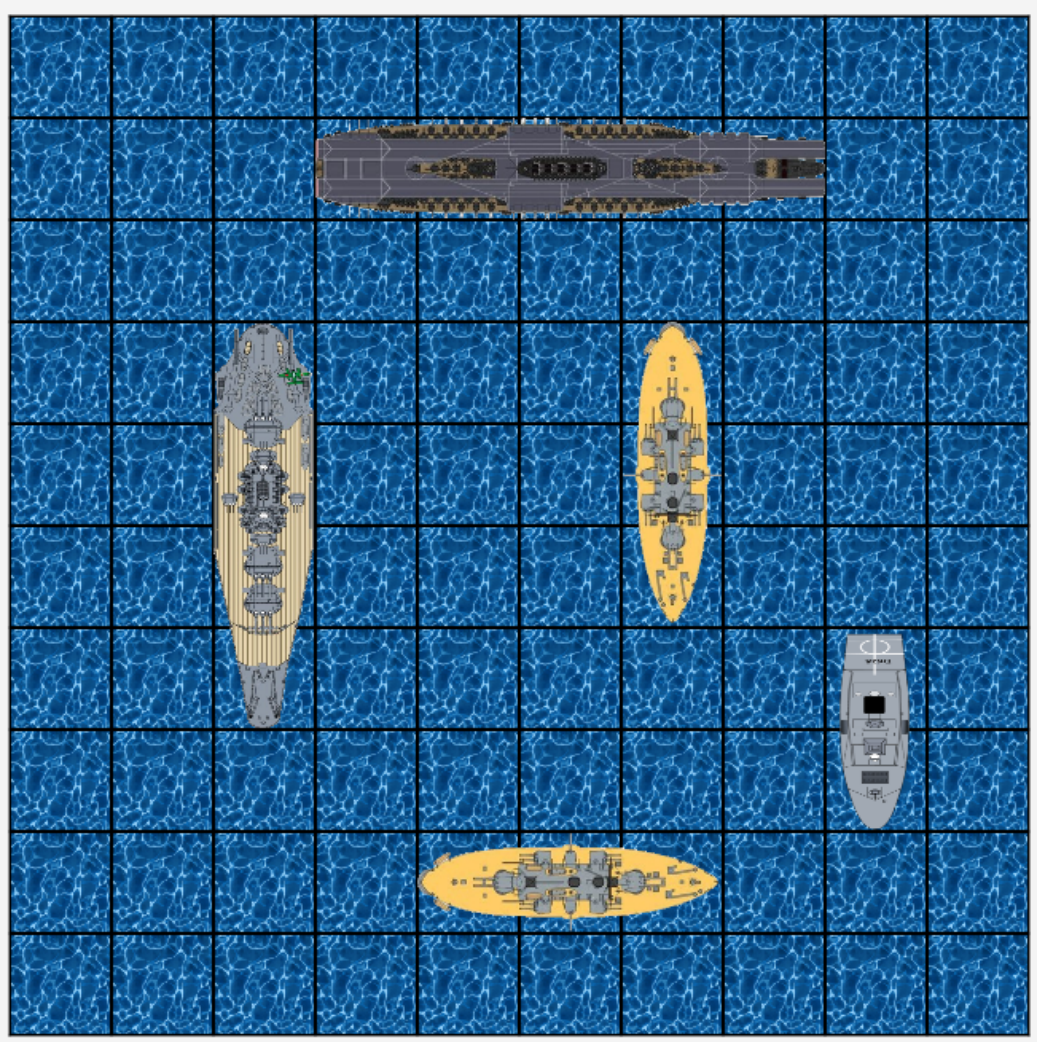

FIGURE 12 – Exemple placement de bateau

Voici ce qui sera envoyé au serveur.

```
3-1,4-1,5-1,6-1,7-1;2-3,2-4,2-5,2-6;6-3,6-4,6-5;4-8,5-8,6-8;8-6,8-7
```

Cette chaîne de caractère est ensuite découpée via des `StringTokenizer` sur le serveur afin de reconstruire le tableau de `Ship`.

### 3.3.4 Sécurité des communication

Afin de garantir une sécurité supplémentaire, les sockets clients et la socket serveur possèdent une couche **SSL**, Secure Sockets Layer. Ce protocole permet la confidentialité des données échangées ainsi que l'intégrité des données échangées.

Pour cela, j'utilise des `SSLSocket` du côté client et une `SSLServerSocket` pour le serveur. Les sockets SSL sont des sockets utilisant une clé, ainsi qu'un certificat auto-signé nommé `keystore.jks` générer via la commande `keytool -genkey -keyalg RSA -alias selfsigned -keystore keystore.jks storepass password -validity 360 -keysize 2048`. Pour pouvoir utiliser cette clé, il a fallu que les sockets aient confiance en ce certificat et cette clé et puissent les utiliser. Cela a mené à l'utilisation de `KeyManager` et de `TrustManager` utilisant l'algorithmes `SunX509`. Le protocole `TLSv1` est utiliser pour créer le `SSLContext` qui sera lui utiliser chez le client et dans le serveur afin de créer les `SSLSocket` et `SSLServerSocket`.

# 4 Interfaces graphiques

Les interfaces graphiques ont été programmées en utilisant JavaFX avec l'utilisation du FXML via l'application 'ScreneBuilder'.

JavaFX est un framework java qui permet de créer des interfaces graphiques pour différents types de supports. JavaFX à pour but de remplacer les bibliothèques awt et swing de java pour pallier les défauts de ces derniers et fournir de nouvelles fonctionnalités (dont le support des écrans tactiles).

Chaque interface est composée de deux parties, un fichier FXML étant la base graphique et un fichier JavaFX se terminant par `Controller` regroupant les méthodes pour pouvoir interagir avec l'interface. Ces fichiers JavaFX possèdent une méthode

```
public void initialize(URL url, ResourceBundle resourceBundle)
```

qui permet d'effectuer des actions sur l'interface comme rajouter des éléments graphiques ou des écouteurs avant l'affichage de ladite interface.

## 4.1 GraphicGrid

`GraphicGrid` est un composant graphique utiliser dans les interfaces graphiques `ShipPlacer` et `InGameAppli`. Il existe deux types de `GraphicGrid`, une ou chaque case de la grille n'est simplement qu'une image, cette version est utilisée pour la grille dans `ShipPlacer` et pour la grille du joueur dans `InGameAppli`. La seconde version et une version ou les cases sont des boutons ce qui permet de créer de l'interactions avec la grille, c'est cette version qui est utilisée pour la grille de l'adversaire dans `InGameAppli`.

`GraphicGrid` possède une méthode qui met à jour l'apparence de grille selon une matrice de `TileState`.

```
public void updateGrid(TileState[][] tab)
```

## 4.2 GraphicShip

`GraphicShip` est un composant graphique utiliser dans `ShipPlacer`. Il permet l'affichage et la manipulation d'un bateau.

Il existe 4 types de bateaux dépendant de la longueur

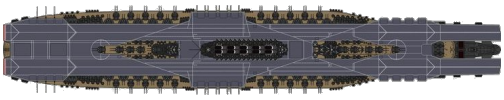

FIGURE 13 – Bateau de taille 5

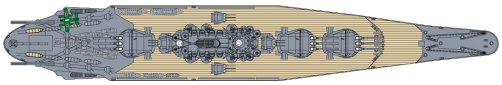

FIGURE 14 – Bateau de taille 4

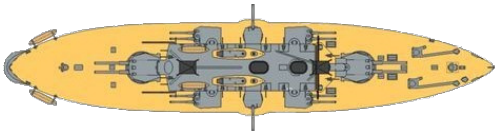

FIGURE 15 – Bateau de taille 3

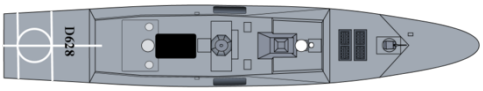


FIGURE 16 – Bateau de taille 2

Pour l'affichage du bateau on utilise une **ImageView**. C'est elle qui est manipulée et c'est sur elle que la classe qui utilise un `GraphicShip` met des écouteurs. Les méthodes de `GraphicShip` sont des méthodes facilitant la manipulation comme :

```
    public void setSelected(boolean selected)

    public void setPlaced(boolean placed)

    public void rotate()
```

- `setSelected` : Permet de dire si le bateau est celui qui est manipulé.
- `setPlaced` : Permet de dire si le bateau est considéré comme placé.
- `rotate` : Permet de mettre le bateau à la verticale ou à l'horizontale.

## 4.3 StartMenuAppli

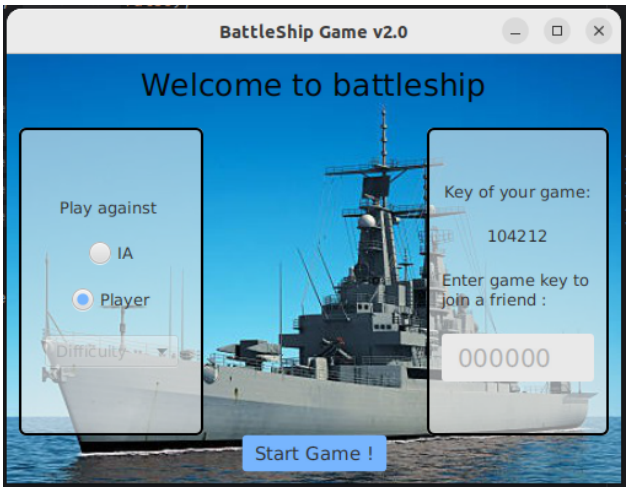


FIGURE 17 – Écran de début de partie V2

Selon le mode de jeu choisi certaine partie de l'interface deviennent inaccessible. Par exemple lorsque l'on sélectionne le mode JvJ on ne peut pas choisir la difficulté de L'IA, car la **ComboBox** est désactivé. Le **TextField** lui est soumis à un pattern matching basé sur l'expression régulier suivante [0-9]6.

## 4.4 ShipPlacer

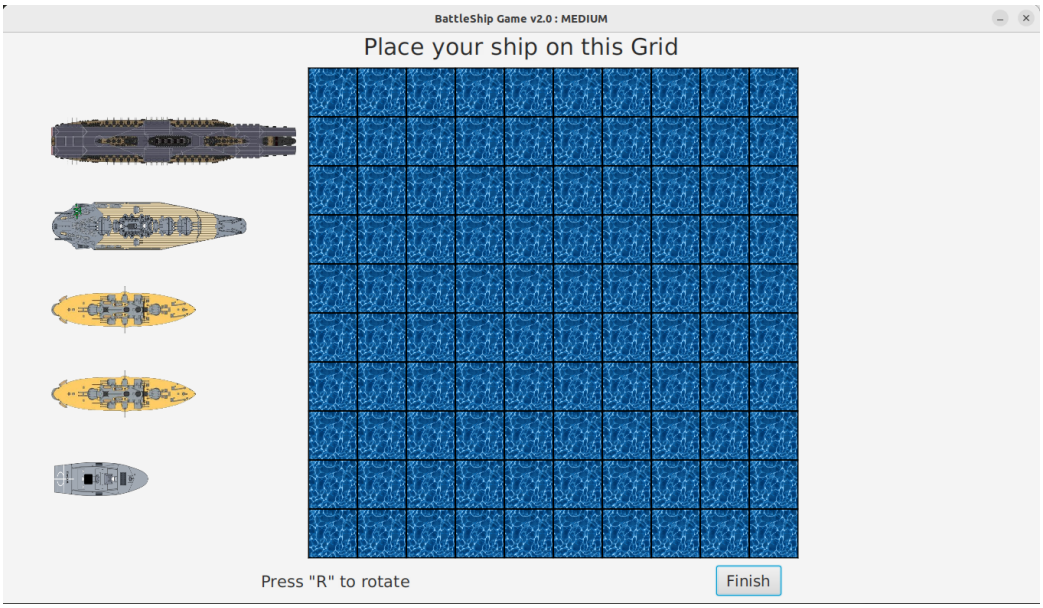


FIGURE 18 – Écran de placement de bateau

La grille est une `GraphicGrid` et chaque bateau est un `GraphicShip`. C'est la classe `ShipPlacerController` qui gère l'ensemble des écouteurs des éléments de cette interface. Les coordonnées d'un bateau sont calculées lors du relâchement de la souris. Si les coordonnées ne sont pas valides alors le bateau est renvoyé sa position de base dans l'interface. De même si le bateau est par-dessus un autre bateau déjà placé.

## 4.5 Écran d'attente

Si vous jouez contre un autre joueur et que vous êtes le premier à se connecter à la partie vous verrez un écran d'attente jusqu'à ce que l'adversaire se connecte.

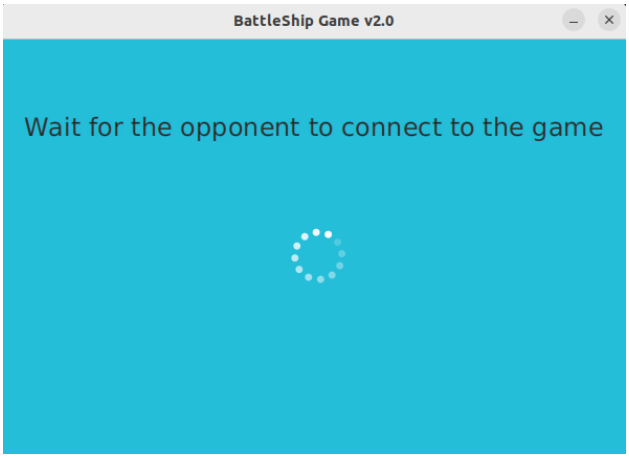


FIGURE 19 – Écran d'attente

## 4.6 InGameAppli

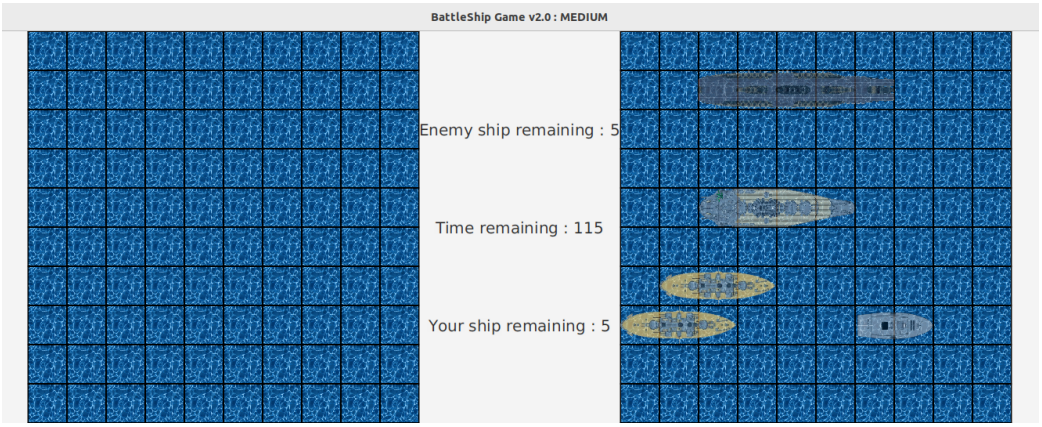


FIGURE 20 – Écran de partie

Les deux grilles sont des `GraphicGrid` et à des fins de confort de jeux les bateaux fantômes représente l'emplacement des bateaux du joueur. Lorsque c'est notre tour, on doit simplement cliquer sur la case que nous souhaitons toucher. À noter que nous ne pouvons toucher deux fois la même case si nous essayons cette pop-up apparaît.

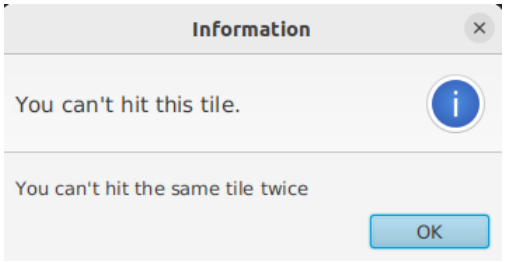


FIGURE 21 – Pop-up en cas de double touche au même endroit

Lorsque c'est le tour de l'adversaire, un effet grisé apparaît sur la grille et aucune action dessus est possible.

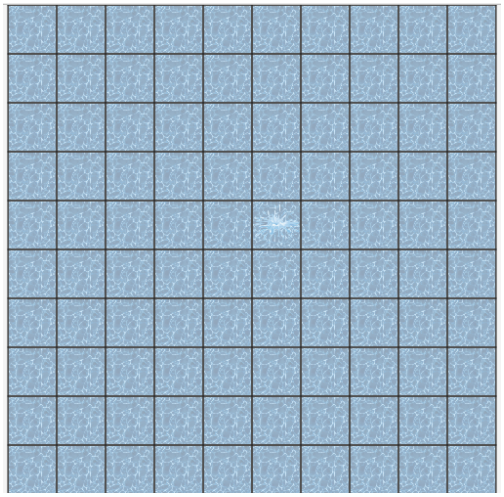

FIGURE 22 – Aspect de la grille de l'adversaire lors de son tour

La mise à jour des labels ainsi que des deux grilles se fait via l'écoute de **PropertyChangeListener** sur les propriétés

- **PROP_TIMER** : Le temps restant pour le tour du joueur.
- **PROP_ENEMY_SHIP** : Le nombre de bateaux ennemis restant.
- **PROP_SHIP** : Le nombre de bateaux restant au joueur.
- **PROP_ENEMY_GRID** : La grille de l'adversaire.
- **PROP_GRID** : La grille du joueur.

D'autre **PropertyChangeListener** sont écoutés tels que

- **PROP_ROUND_PLAYER** : Avoir connaissance si c'est le tour du joueur ou non.
- **PROP_LOSE_GAME** : Savoir si la partie est perdu.
- **PROP_WIN_GAME** : Savoir si la partie est gagnée.

## 4.7 Écran de fin de partie


FIGURE 23 – Pop-up de fin en cas de victoire


FIGURE 24 – Pop-up de fin en cas de défaite

Lors de la fin d'une partie on a le choix entre redémarrer une partie **Restart**, ce qui nous amène vers l'écran de démarrage du jeu. Le bouton **Quit** qui quitte l'application et le bouton **OK** qui vas simplement fermer cette fenêtre tout en laissant l'interface de **InGameAppli** inchangé.

# 5 Annexe

## 5.1 Utilisation du programme

### 5.1.1 Programme Client

- Exécuter le fichier `launch.sh` dans le même répertoire que le fichier `keystore.jks`.
- Ou éxécuter cette commande :

```
java --module-path ./javafx-path/lib
--add-modules javafx.controls,javafx.fxml,javafx.media
-jar BatailleNavale-1.0.jar
```

Le jar doit être dans le même répertoire que le fichier `keystore.jks`.

### 5.1.2 Programme Serveur

- Exécuter `java -jar ServerBatailleNavale.jar` dans un terminal dans le même répertoire que le fichier `keystore.jks`.

### 5.1.3 Déroulement d'une partie en ligne

Voici les étapes à suivre pour démarrer et jouer une partie contre un autre joueur en réseau :

- Dans l'écran de démarrage, cochez la case 'Player', puis dans le champ de texte 'key', rentrez un identifiant de partie (où laisser vide pour utiliser celui générer aléatoirement afficher au-dessus). Une fois fait, cliquez sur le bouton 'Start Game' pour lancer la partie.

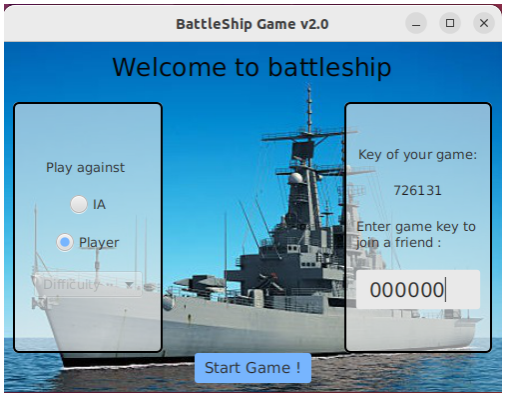


FIGURE 25 – Utilisation du menu start contre un joueur

- Une fois sur l'écran de positionnement des bateaux, sélectionne un par un vos bateaux et placez-les sur la grille en maintenant le clique gauche enfoncée. Pour tourner le bateau un simple clic sur la touche 'R' du clavier suffit.

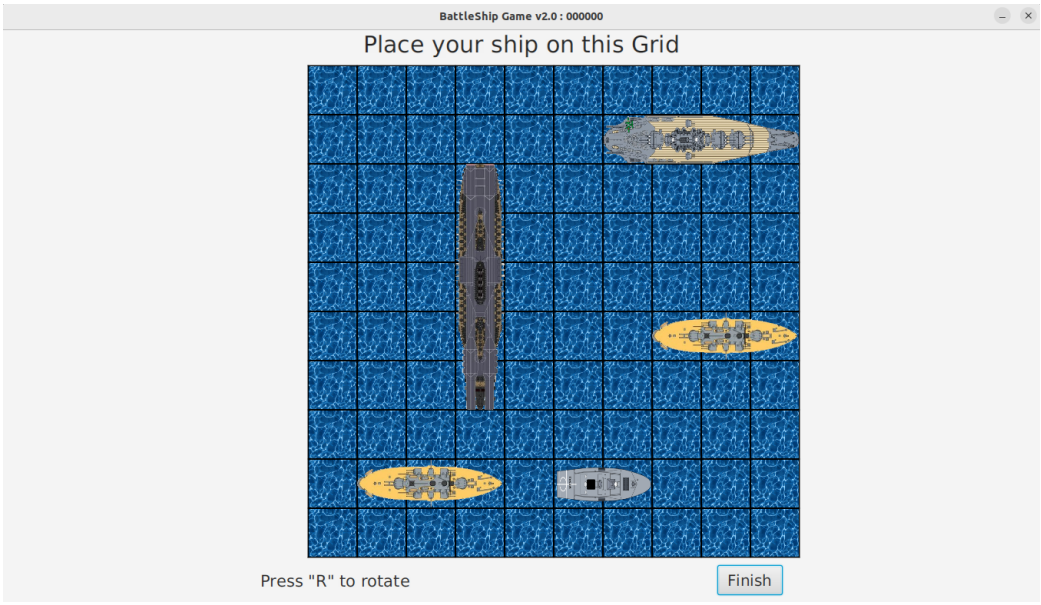


FIGURE 26 – Utilisation de l'application de placement de bateau

- Maintenant que tous vos bateaux sont positionnés, appuyez sur le bouton 'Finish' pour passer à la suite.
- Vous pouvez maintenant jouer en tour par tour à la bataille navale, il suffit de cliquer sur la case que vous souhaitez toucher.

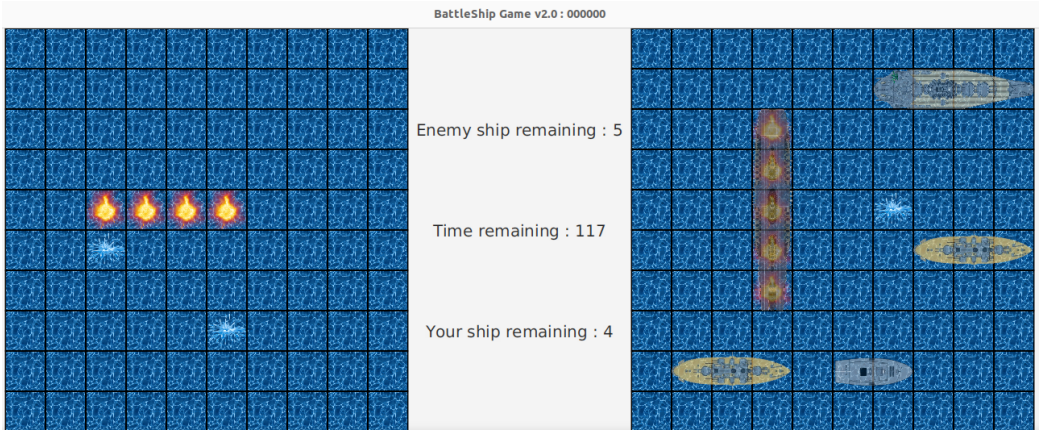


FIGURE 27 – Utilisation de l'application de jeux

- Une fenêtre apparaîtra dès que la partie sera terminée pour vous informer que vous avez gagné, perdu ou pire encore, que le serveur a eu un problème.


FIGURE 28 – Exemple de fin de partie où le joueur gagne

## 5.2 Difficultés rencontrées

- La première difficulté que j'ai rencontrée et qui a duré toute la durer du projet est liée au fait que je découvrais JavaFX en même temps que je programmais ce qui à mener à passer beaucoup de temps à faire des recherches et aussi à beaucoup d'erreurs ou de parties non optimisées menant à des problèmes lorsque l'on veut modifier des parties ou régler des problèmes.
- La deuxième difficulté était sur l'interface graphique `ShipPlacer` pour la partie de 'drag-and-drop' des bateaux, car il fallait que sur la grille les bateaux se déplace case par case et qu'on ne puisse pas mettre un bateau à moitié sur la grille et à moitié dehors, mais il fallait que tous, ces choses sois aussi respectés lorsque l'on tourne le bateau.
- La dernière difficulté est liée à la deuxième, c'est la gestion de la rotation des `GraphicShip` car la manière dont je l'ai fait ne permet pas une très grande liberté de manipulation, mais étant donné que c'est l'unique moyen que j'ai trouvé, c'est celui que j'ai adopté.

## 5.3 Amélioration globale possible

Des améliorations sur l'ensemble du projet possibles. En termes de fonctionnalités données aux joueurs on peut penser à :

- Le choix du nombre de bateaux : Cette fonctionnalité peut être introduite de plusieurs manières : Sois les joueurs choisis le nombre de

chaque bateau ou avec une fonction qui génère automatiquement le bon nombre de chaque bateau en fonction du nombre total de bateaux. Les limites de cette fonctionnalité ont aussi plusieurs façons d'être introduites. Sois avec un nombre maximum défini en dur dans le code via une macroconstante ou avec une fonction qui calcule la totalité des cases occupées par les bateaux et si on peut donc placer l'ensemble des bateaux sur la grille.

- GraphicShip : le composant graphique pour représenter les bateaux peut très sûrement être fait d'une autre manière permettant plus de liberté et limitant le nombre de problèmes pouvant être causé par la limitation de l'implémentation actuelle.

# 6 Conclusion

Avoir retravaillé sur ce projet m'as permis de découvrir de nouvelles choses telles que JavaFX et FXML. Ainsi que m'améliorer sur mon point faible, les interfaces graphiques en Java. Je remercie mes camarades avec lesquelles j'ai fait ce projet dans le cadre de l'Application Informatique, car j'ai pu réutiliser les travaux ou m'en inspiré.

Je remercie également M. Guesnet pour ces conseils, avis et idées sur le projet.

Je vous remercie d'avoir lu mon rapport, Leroy Florent.